Hindawi
Mobile Information Systems
Volume 2022, Article ID 9587040, 13 pages
https://doi.org/10.1155/2022/9587040

Research Article

GMOM: An Offloading Method of Dependent Tasks Based on Deep Reinforcement Learning

Huiji Zheng,[1] **Sicong Yu**,[1] **and Xiaolong Cui**[2]

[1]*School of Information Engineering, Engineering University of PAP, Xi'an 710086, China*
[2]*Counter-Terrorism Command Information Engineering Research Team, Engineering University of PAP, Xi'an 710086, China*

Correspondence should be addressed to Huiji Zheng; 1344261395@qq.com

Received 27 May 2022; Revised 15 August 2022; Accepted 28 October 2022; Published 8 November 2022

Academic Editor: Adrian Kliks

Mobile edge computing (MEC) is considered as an effective solution to delay-sensitive services, and computing offloading, the central technology in MEC, can expand the capacity of resource-constrained mobile terminals (MTs). However, because of the interdependency among applications, and the dynamically changing and complex nature of the MEC environment, offloading decision making turns out to be an NP-hard problem. In the present work, a graph mapping offloading model (GMOM) based on deep reinforcement learning (DRL) is proposed to address the offloading problem of dependent tasks in MEC. Specifically, the MT application is first modeled into a directed acyclic graph (DAG), which is called a DAG task. Then, the DAG task is transformed into a subtask sequence vector according to the predefined order of priorities to facilitate processing. Finally, the sequence vector is input into an encoding-decoding framework based on the attention mechanism to obtain the offloading strategy vector. The GMOM is trained using the advanced proximal policy optimization (PPO) algorithm to minimize the comprehensive cost function including delay and energy consumption. Experiments show that the proposed model has good decision-making performance, with verified effectiveness in convergence, delay, and energy consumption.

1. Introduction

With the development of Internet of things (IoT) and mobile computing, mobile terminals and applications, such as virtual reality (VR) and facial recognition applications, are seeing increased diversity and complexity [1, 2]. Despite the increasingly improved performance of mobile terminals, many computing-intensive applications cannot be processed efficiently and delay results in compromised user experience and poorer service quality. In this case, mobile edge computing (MEC) comes as a solution. An emerging computing mode, MEC is born of traditional cloud computing. Its central goal is to expand the rich resources from the cloud server to the user terminals so that the user can employ more abundant resources to process their own computing tasks nearby, thus reducing the delay and energy consumption. Computing offloading, a key technology in MEC, refers to the process in which the mobile terminal assigns computing-intensive tasks to edge servers with sufficient computing resources through the wireless channel according to a certain strategy, and then, the edge servers return the computation results to the mobile terminal [3]. The computing offloading problem can be transformed into an optimization problem in a specific environment, but due to the complexity of the MEC environment, it is challenging to address this problem by traditional approaches [4].

In practice, many computing tasks are not completely independent from each other, but involve multiple subtasks (such as VR/AR and face recognition). The input of some tasks comes from the output of other tasks, that is, some tasks need to be processed before the current task can be executed. If the dependency between the tasks is not considered, the application may fail to run properly. The offloading of dependent tasks is generally modeled as a DAG to achieve fine-grained task offloading [5]. However, as the number of task nodes increases, it is difficult to obtain the optimal offloading strategy for all subtasks [6]. Most existing works in this regard use heuristic or metaheuristic algorithms [7–10]. Specifically, in order to minimize the application delay, reference [7] designs a heuristic DAG decomposition scheduling algorithm, which considers the load balance between user devices and servers. In reference [8], a heuristic algorithm is proposed to solve the task offloading problem of mobile applications in three steps to minimize the system energy consumption under delay constraint. Reference [9] considers that some tasks in DAG can only be executed locally, so as to minimize the waiting delay among tasks executed locally. This problem is modeled as a nonlinear integer programming problem and approximated by a metaheuristic algorithm. However, these methods cannot fully adapt to dynamic MEC scenarios because of the contradiction between flexibility and computational cost when designing heuristics [10].

Boasting the strengths of both deep learning and reinforcement learning, DRL has strong perception and decision-making ability, so it has been widely studied and applied [11] to complex decision-making problems with high-dimensional state/action spaces, such as games [12] and robots [13]. Through continuous interaction with the environment, the DRL model learns appropriate strategies (what actions to perform in a given environment) with the goal of maximizing long-term returns. Considering indivisible and delay-sensitive tasks and edge load dynamics, a model-free distributed algorithm based on DRL is proposed in reference [14]. Each device can determine its offloading decision without knowing the task model and offloading decision of other devices. In order to cope with the challenge of task dependence and adapt to dynamic scenarios, a new DRL-based offloading framework is proposed in reference [15], which can effectively learn the offloading strategy uniquely represented by a specially designed sequence-to-sequence neural network. Considering the limited performance of a traditional single-type offloading strategy in a complex environment, reference [16] designs a dynamic regional resource scheduling framework based on DRL, which can effectively consider different indexes. Mobile edge computing with the energy harvesting function is considered in reference [17]. In order to solve the challenge of coordination between continuous and discrete interaction space and devices, two dynamic computing offloading algorithms based on DRL are proposed. Simulation results show that the proposed algorithm achieves a better balance between delay and energy consumption.

In the present work, a graph mapping offloading model based on deep reinforcement learning is proposed. First, the mobile terminal is modeled as a directed acyclic graph (DAG). Then, the DAG task is transformed into a subtask sequence vector in order of priority. Finally, the policy function based on the recurrent neural network (RNN) is input to obtain the offloading strategy. The proposed model employs the proximal policy optimization (PPO) algorithm to minimize the comprehensive cost function including delay and energy consumption. The major contributions of this paper are as follows:

(1) Considering the inherent task dependency of mobile applications, we innovatively propose a DRL-based task offloading model, which leverages off-policy reinforcement learning with RNN to capture dependencies.

(2) We design a new encoding method to encode DAG into task vector, including task profile and dependency information, which can transform DAG into RNN input without loss of fidelity.

(3) We introduce an attention mechanism to solve the problem of performance degradation caused by the incomplete capture of feature information in long sequences. To effectively train the model, we use an off-policy DRL algorithm with a clipped surrogate objective function to make the algorithm have strong exploration ability and prevent the training from getting stuck in the local optima.

2. System Model

Figure 1 shows the MEC scenario considered in the present work. The mobile terminal (MT), as the end user, is the service requester and generates computing tasks according to the program (path planning and object identification) called by the user. These applications have internal dependency, so in the present work, computing offloading is defined as sequential offloading of subtasks. Subtasks are transmitted to the edge base station (BS) for execution. Finally, the processed result will be sent back to the MT. The system model of this work will be described in detail.

2.1. Task Model.

Each MT generates a computing-intensive application with N tasks, as shown in Figure 2. We modeled the application as a DAG, and $G = (V, E)$; the vertex set $V = \{v_1, v_2, \ldots, v_N\}$ represents each computation task; and the edge set $E = \left\{e = (v_i, v_j) | (i, j) \in (1, 2, \ldots N) \times (1, 2, \ldots N)\right\}$ represents the dependency between computation tasks, where v_i is a direct predecessor to v_j, and v_j is a direct successor to v_i. In particular, a task without a predecessor task is called an *entry* task, and a computing task without a successor task is called an *exit* task [18].

Each computing task in G is represented by a tuple $v_i = (c_i, d_i, q_i)$, which represents the number of CPU cycles required, the size of input and that of output data, respectively. The input data generally include the source codes and related parameters in the application program, while the output data refer to the data generated by the predecessor task [19]. Each task has two computing modes as follows: the task is processed on MT (which is termed local computing) and then transmitted to the base station (BS) for processing through the wireless channel before being returned to MT, i.e., offloading computing. The computation mode of all tasks can be represented as a list $A = (a_1, a_2, \ldots, a_N)$ and $a_i \in \{0, 1\}$, in which 0 indicates local computing and 1 indicates offloading computing. When the computation tasks are intensive, since the execution of task v_i may need to queue up in the task queue, and considering the restriction of dependency relationship, the execution of task v_i needs to meet two conditions as follows:

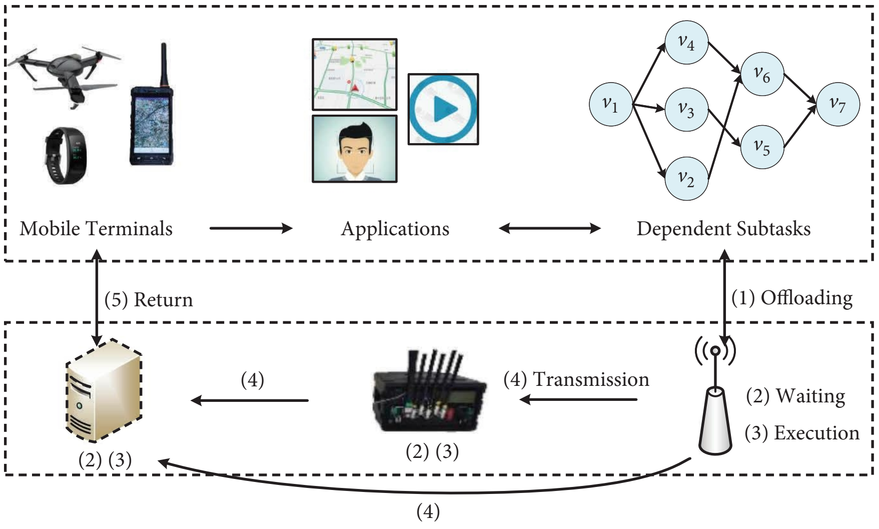


Figure 1: MEC scenario.

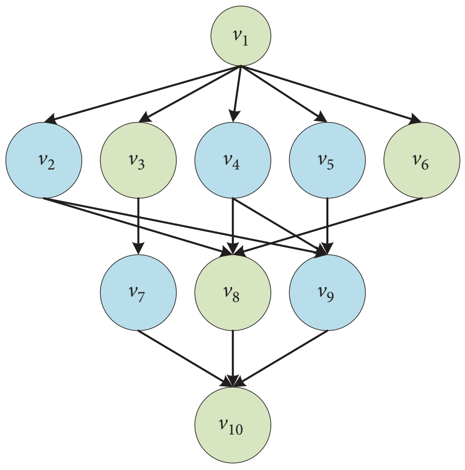


Figure 2: Structure of a directed acyclic graph (DAG).

(1) all predecessors to task v_i have been completed

(2) the resources required for task v_i are idle.

In order to better represent the computing model and communication model, the following definitions are first given:

Definition 1. The completion time of task v_i in the wireless uplink and downlink channel is expressed as FT_i^s and FT_i^r, respectively, and the completion time of task v_i on MT and BS is expressed as FT_i^l and FT_i^o, respectively.

Definition 2. The available time that task v_i can use the wireless uplink and downlink channels is denoted as AT_i^s and AT_i^r, respectively, and the available time of task v_i can use the computing resources on MT and BS is expressed as AT_i^l and AT_i^o, respectively.

The available time represents when the resource is idle and depends on the completion time of the immediate precursor task of task v_i on that resource. If the precursor task does not utilize the resource, we set the completion time on that resource to 0.

2.2. Local Computing Model. If v_i is computed locally, the start time of task v_i depends on the completion time of its precursor task and the available time of MT. The precursor task may be computed in the MT or BS. In this case, the completion time in MT is equal to the start time plus the local execution delay.

$$\mathrm{FT}_i^l = \max\left\{\max\left\{\mathrm{FT}_p^l, \mathrm{FT}_p^r\right\}, \max\left\{\mathrm{AT}_p^l, \mathrm{FT}_p^l\right\}\right\} + T_i^l, \tag{1}$$

where v_p is the direct predecessor task of v_i. If v_p is computed locally, $\mathrm{FT}_p^r = 0$; if v_p is computed remotely, $\mathrm{FT}_p^l = 0$. While MT is processing other tasks, v_i needs to queue and cannot be executed immediately. The local execution delay of v_i is $T_i^l = d_i/f^l$, where f^l represents the computing capacity of MT. Meanwhile, the energy consumption is $E_i^l = k \cdot c_i \cdot (f^l)^2$, where k is dependent on the effective capacitance coefficient of the chip structure used.

2.3. Offloading Computing Model. If v_i chooses offloading computing, MT first needs to send v_i to BS through the wireless uplink channel, but it can only be sent when all the precursor tasks have been executed and the uplink channel is available; then, the completion time of the wireless uplink channel is equal to the start time plus the sending delay.

$$\mathrm{FT}_i^s = \max\left\{\max\left\{\mathrm{FT}_p^l, \mathrm{FT}_p^r\right\}, \max\left\{\mathrm{AT}_p^s, \mathrm{FT}_p^s\right\}\right\} + T_i^s. \tag{2}$$

According to Shannon's theorem [20], the uplink rate from MT to BS is as follows:

$$r_{\mathrm{up}} = w \cdot \log_2\left(1 + p \cdot \frac{g}{\sigma^2}\right), \tag{3}$$

where w, p, g, and σ^2 represent system bandwidth, transmission power of MT, channel gain, and noise power, respectively. We assume that the uplink rate and downlink rate of a wireless channel are equal, then $r_{\text{down}} = r_{\text{up}}$. The transmission delay of MT sending data to BS is $T_i^s = d_i/r_{up}$, and the corresponding energy consumption is $E_i^s = p^s \cdot T_i^s$, where p^s represents the transmission power.

After BS receives v_i, if the conditions are met, it can execute v_i. In this case, similarly, the execution completion time in BS is equal to the start time plus the execution delay on BS.

$$\mathrm{FT}_i^o = \max\left\{\max\left\{\mathrm{FT}_i^s, \max \mathrm{FT}_p^o\right\}, \max\left\{\mathrm{AT}_p^o, \mathrm{FT}_p^o\right\}\right\} + T_i^o. \quad (4)$$

The delay of execution of v_i in BS is $T_i^o = c_i/f^s$, where f^s represents the computing capacity of BS. Similarly, the completion time of BS sending the processing results back to MT through the wireless downlink channel is as follows:

$$\mathrm{FT}_i^r = \max\left\{\mathrm{FT}_i^o, \max\left\{\mathrm{AT}_p^r, \mathrm{FT}_p^r\right\}\right\} + T_i^r. \quad (5)$$

The return delay is $T_i^r = q_i/r_{\text{down}}$ and the energy consumption is $E_i^r = p^r \cdot T_i^r$.

2.4. Problem Description. According to the abovementioned analysis, the delay of executing a DAG task is as follows:

$$T_A = \max_{v_i \in K}\left\{\max\left\{\mathrm{FT}_i^l, \mathrm{FT}_i^r\right\}\right\}, \quad (6)$$

where K is the set of all *exit* tasks, and the sum delay is equal to the time required to process till the last *exit* task.

The energy required to consume is as follows:

$$E_A = \sum_{i=1}^{N} E_i = \sum_{i=1}^{N}\left(E_i^l \cdot \mathrm{I}_{(a_i=0)} + E_i^o \cdot \mathrm{I}_{(a_i=1)}\right), \quad (7)$$

where $E_i^o = E_i^s + E_i^r$ represents the energy consumption required for offloading computing, $\mathrm{I}_{(\Delta)}$ is an indicator function whose value is equal to 1 when condition Δ is satisfied, otherwise, its value is 0.

To measure the quality of an offloading strategy, we define a comprehensive cost function [21, 22], and the ultimate goal is to minimize the comprehensive cost value.

$$\mathrm{CC} = \alpha \cdot T_A + (1-\alpha) \cdot E_A. \quad (8)$$

where T_A and E_A represent the required delay and energy consumption, respectively, and α is the balance factor, which is valued in the interval [0, 1]. This weighted sum method is effective and easy to implement, so it has been widely used. The balance factor reflects that user preferences can be adjusted dynamically. However, the offloading problem of general DAGs is NP-hard [23], so it is difficult to find an optimal offloading strategy with appropriate time complexity.

3. Graph Mapping Offloading Model Based on DRL

This section mainly introduces in detail the graph mapping offloading model based on DRL. First, to facilitate the processing of DAG tasks, the DAG tasks are converted into subtask sequence vectors by the heterogeneous earliest finish time (HEFT) algorithm. This step acts as a data preprocessing step, after which the subtasks are processed with reference to the vector, and the computing process is detailed through an example. Then, the offloading problem is described as a Markov decision process (MDP), and the corresponding state space, action space, and reward are analyzed. Finally, the structure of the graph mapping offloading model is introduced, which is based on an encoding-decoding framework with the attention mechanism and trained by the proximal policy optimization (PPO) algorithm.

3.1. DAG Instance. In order to satisfy the dependency constraint between subtasks in the application program, we prioritize tasks in descending order based on their sequence values, which is defined as follows:

$$\mathrm{rank}(v_i) = \begin{cases} T_i^o, & \text{if} v_i \in K, \\ \max_{v_j \in \mathrm{succ}(v_i)}\left\{\mathrm{rank}(v_j) + T_i^o\right\}, & \text{if} v_i \notin K. \end{cases} \quad (9)$$

The priority is calculated in a similar way to the HEFT algorithm [24], where $succ(v_i)$ represents the direct successor task set of task v_i, T_i^o is the execution cost of task v_i, which can be calculated as follows: $T_i^o = T_i^s + T_i^b + T_i^r$, the shortest delay (without queuing) obtained by offloading the task. We treat the sort order as an execution order vector, represented by $O = [o_1, o_2, \ldots, o_n]$, where o_i represents the ith task to be executed. Table 1 shows the execution delays of tasks on the MT and BS. According to formula (9), the execution order vector $O = [v_1, v_2, v_4, v_5, v_3, v_6, v_9, v_8, v_7, v_{10}]$ can be obtained.

The DAG in Figure 2 is used as an example, and we briefly illustrate the execution of the task. This application program G is composed of 10 subtasks. It is assumed that the offloading strategy vector is $A = [0, 1, 1, 1, 0, 0, 1, 0, 1, 0]$, and Figure 3 shows the task execution process based on O and A. For example, if task v_5 is offloaded to the BS, its available time is $\mathrm{AT}_9^s = 6\mathrm{s}$ and its completion time is $\mathrm{FT}_9^r = 14\mathrm{s}$. From Figure 3, we know that the execution latency of DAG task is $T_A = 20\mathrm{s}$.

3.2. Structure of the Markov Decision Process. Each task is connected to a virtual machine to provide private computing, communication, and storage resources, so the parameters of the network environment are considered to be static. Therefore, we establish MDP from the perspective of DAG task.

State space: The state space is a sequence of information about DAG tasks and offloading decisions of historical subtasks. The policy set of the history subtask is represented by $A_{1\sim i}$, then

$$\text{state} = \left\{G, A_{1\sim i}\right\}. \quad (10)$$

Action space: Since each subtask either selects local computing or offloading computing, the action space is expressed as follows:

Table 1: Task execution delay(s).

Task	T_i^l	T_i^s	T_i^b	T_i^r
v_1	2	1	1	1
v_2	6	1	3	1
v_3	4	2	2	1
v_4	2	2	1	2
v_5	2	1	1	1
v_6	4	2	2	1
v_7	8	1	4	1
v_8	2	2	1	1
v_9	6	4	3	1
v_{10}	2	1	1	1

$$\text{action} = \{0, 1\}. \tag{11}$$

Reward: When task v_i is completed according to strategy a_i, the total delay increment is $\triangle T_i$ and the required energy consumption is E_i. Therefore, the reward function can be defined as follows:

$$\begin{aligned} r_i^T &= -\triangle T_i, \\ r_i^E &= -E_i. \end{aligned} \tag{12}$$

3.3. Model Design. The graph mapping offloading model is based on an encoding-decoding framework that incorporates the attention mechanism [25]. The underlying idea of the model is to use two RNNs, one as an encoder and the other as a decoder. The task vector is input into the encoder, where the feature information of the task is extracted; then, feature decomposition is performed through the decoder and the offloading decision is output. However, this simple encoding-decoding framework is instable in remembering previous feature information in long input sequences, which may lead to poor performance of the model. The attention mechanism is hence introduced in our work to solve this problem. The attention mechanism can reduce the information loss in the simple encoding-decoding framework by calculating the correlation between the hidden state of each step in the decoder and the hidden state of each step in the encoder and by assigning a corresponding weight value to the extracted features. As shown in Figures 4 and 5, the specific process is as follows.

At time step i, the encoder transforms the input task vector v_i and the hidden state h_{en}^{i-1} of the last time step into the hidden state h_{en}^i of the current time step, namely, $h_{en}^i = f_{en}(v_i, h_{en}^{i-1})$, where f_{en} represents the encoder network. Then, the weight of the hidden states of each time step of the encoder is calculated, and $c_j = \sum_{i=1}^N \xi_{ji} h_{en}^i$ is obtained to form a context vector $C = [c_1, c_2, \ldots, c_N]$, where ξ_{ji} represents the weight of each hidden state of the encoder, which is normalized to the alignment vector by SoftMax operation. In each time step j of decoding, the hidden state h_{de}^j of the decoder at this step is not sent to the output layer at first but calculates the score with each hidden state of the encoder, that is, the alignment vector $\text{align}_j = [\text{score}(h_{de}^j, h_{en}^1), \ldots, \text{score}(h_{de}^j, h_{en}^i)]$ is obtained. In the present work, the score function is defined as a trainable neural network [26], which represents the correlation between the hidden state of each time step in the decoder and the hidden state of each step in the encoder.

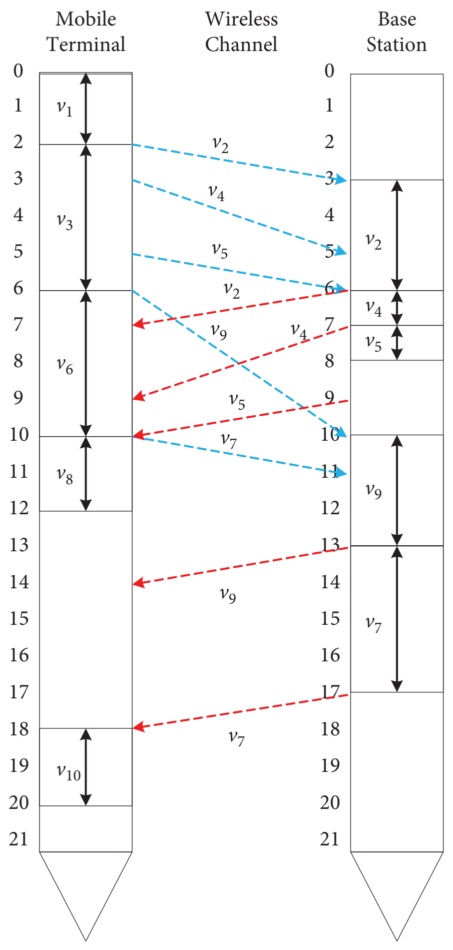


Figure 3: An example of DAG task execution.

At the time step j, the decoder transforms the corresponding context value c_j, the last hidden state h_{de}^{j-1}, and the last output result a_{j-1} into the current hidden state $h_{de}^j = f_{de}(h_{de}^{j-1}, a_{j-1}, c_j)$. In the final output, the decoder obtains the strategy $\pi(a_j | s_j; \omega)$ through the SoftMax layer and selects the action through $a_j = \text{argmax}\pi(a_j | s_j; \omega)$.

The state value $V(s_t; \theta)$ can be calculated by the critic network, where θ is the parameter of the critic network. The network is composed of a recurrent neural network and a full connection layer, which is initialized by the final hidden state of the encoder in the actor network. A history offloading policy whose final hidden state is mapped to a state value through the full connection layer is entered.

3.4. Model Training. PPO is a reinforcement learning algorithm based on the actor-critic (AC) framework proposed by OpenAI, which mainly consists of three networks: one critic network and two actor networks (actor network and old actor network). The old actor network is used to generate training data, while the actor network uses the generated training data for training. Compared with the previous trust

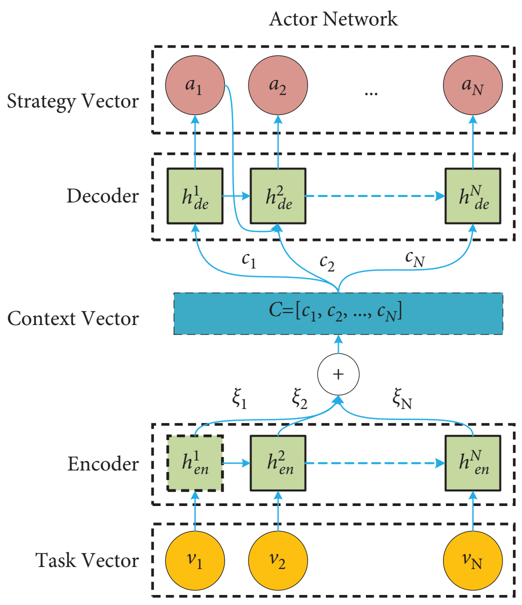


Figure 4: Actor network in GMOM.

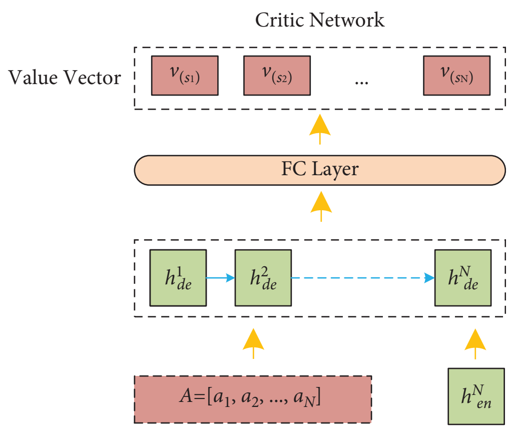


Figure 5: Critic network in GMOM.

region policy optimization (TRPO) algorithm, PPO is easier to implement and has good performance, which solves the problem that the policy gradient algorithm is difficult to determine the step size and the update difference is too large [27]. In order to control the updated step of the policy, PPO adopts the clipped surrogate objective function, as shown in the following formulae:

$$L_t^{\mathrm{CLIP}}(\theta) = E\left[\sum_{t=1}^{N} \min\left(pr_t(\theta)A_t(s_t, a_t), \operatorname{clip}(\cdot)A_t(s_t, a_t)\right)\right],$$

$$\operatorname{clip}\left(pr_t(\theta), 1-\varepsilon, 1+\varepsilon\right), \tag{13}$$

$$pr_t(\theta) = \frac{\pi\left(a_t|G, A_{1\sim N}; \theta\right)}{\pi\left(a_t|G, A_{1\sim N}; \theta_{\mathrm{old}}\right)}.$$

The clip function clip$(\cdot)$ aims to limit the value of the importance of sampling weight $pr_t(\theta)$, where ε is the hyperparameter controlling the clip range. The min function minimizes the original item and the truncated item, that is, the truncated item will limit its value when the offset of the policy update exceeds the predetermined interval.

The actor network and the critic network in the PPO algorithm share network parameters, and its training and optimization objective function is defined as follows:

$$L_t^{\mathrm{PPO}}(\theta) = E\left[L_t^{\mathrm{CLIP}}(\theta) - c_1 L_t^{\mathrm{VF}}(\theta) + c_2 S(s_t)\right], \tag{14}$$

where $L_t^{\mathrm{CLIP}}(\theta)$ and $L_t^{\mathrm{VF}}(\theta)$ are the loss functions of the actor network and the critic network, respectively; $S(s_t)$ is the cross entropy loss, which is used to improve the strategy exploration ability; c_1 and c_2 are constants. The PPO algorithm is based on an actor-critic (AC) architecture, so it needs to optimize these two groups of parameters, respectively.

For the actor network, general advantage estimation (GAE) is used as an estimation function of their loss functions to balance variances and bias. It borrows the idea of the time-series difference algorithm and is expressed as follows:

$$A_t^{GAE(\gamma,\lambda)} = \sum_{k=0}^{N-t+1} (\gamma\lambda)^k \delta_{t+k}^V, \tag{15}$$

where $A_t^{\mathrm{GAE}(\gamma,\lambda)}$ is the advantage function and δ_{t+k}^V is the timing difference error, and the calculation formula is as follows:

$$\delta_{t+k}^V = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{16}$$

For the critic network, its loss function is expressed as follows:

$$L_t^{\mathrm{VF}}(\theta) = E\left[V(s_t;\theta) - V^{\mathrm{targ}}(s_t)\right]^2, \tag{17}$$

where $V^{t\,\mathrm{arg}}(\cdot)$ is the estimated value of the value function and is defined as follows:

$$V^{\pi^*}(s_t) = \sum_{k=1}^{N-t+1} \gamma^k r_{t+k}. \tag{18}$$

Table 2 shows the training process of the proposed algorithm. The old actor network is used for sampling and the actor network for training. The proposed algorithm alternates between the exploration stage (lines 5–9) and the optimization stage (lines 10–13). In the exploration stage, D episodes are collected in each time step using the old actor network. GAE estimates $A_t^{\mathrm{GAE}(\gamma,\lambda)}$ for each time step in each episode and the value $V^{\pi^*}(s_t)$ for each state is also calculated. In the optimization stage, the batch sample data are sampled from the experience pool, and the objective function is optimized in round H according to the sampling data. Then, the old actor network is updated with the updated actor network to ensure that the old actor network collects data with the new parameters in the next sampling stage.

4. Simulation Results and Discussion

The simulation experiment is completed on a workstation with Intel Core I7–9700H 3.6 GHz CPU and 8 GB memory. The virtual environment is TensorFlow-GPU-1.x. The

Table 2: Training process of the GMOM model.

```
Algorithm GMOM
(1) Initialize the initial network parameter $\theta$ that the actor network and the critic network share randomly
(2) Initialize the parameter $\theta_{\mathrm{old}}$ of the old actor network with $\theta$
(3) For iteration = 1, 2, ... do
(4)     for $t = 1, 2, \ldots, N$ do
(5)         for $i = 1, 2, \ldots, D$ do
(6)             the whole episode is collected with the old actor network, and the obtained data is stored in the experience pool $D$
(7)             calculate the GAE function value for each time step according to formula (14), get $A_t^{\mathrm{GAE}(\gamma,\lambda)}$ and cache it
(8)             calculate the value in each state according to formula (17) and get $V^{\pi^*}(s_t)$
(9)         end
(10)        for $j = 1, 2, \ldots, H$ do
(11)        sample batch size sample data to optimize the objective function, update the actor network
(12)        end
(13)        Synchronize the parameters of two actor networks, i.e., $\theta_{\mathrm{old}} \leftarrow \theta$
(14)    end
(15) end
```

encoding-decoding framework contains two RNNs, one as an encoder consisting of 256 hidden units and two fully connected layers; the other RNN acts as a decoder and consists of 256 hidden units, 1 full connection layer, and 1 SoftMax layer. Tanh is used as the activation function and Adam as the optimizer. The discount factor γ is set to 0.95. The maximum number of episodes is set to 2000. Other parameters are shown in Table 3.

A popular method is employed here to generate various DAGs to simulate applications with dependencies [28]. The parameters of DAG are shown in Table 4, where fat and de nsity determine the width, weight, and dependency of DAGs, respectively.

Table 3: Parameters of the simulation experiment.

Parameter	Value	Parameter	Value
r_{down}, r_{up}	{4.5, 8.5, 12.5, 16.5} Mbps	σ^2	10^{-6} W
w	0.6 MHz	f^l	0.6 GHz
p^s	1.2 W	f^s	6 GHz
p^r	1.1 W	k	10^{-26}

Table 4: Specifics of DAG instances.

Instance	N	Fat	Density	c_i (cycle/ sec)	d_i (kb)	q_i (kb)
DAG-1	10	0.6	0.3	[10^7, 10^8]	[5, 50]	[0.5, 5]
DAG-2	10	0.6	0.7	[10^7, 10^8]	[5, 50]	[0.5, 5]
DAG-3	20	0.6	0.7	[10^7, 10^8]	[5, 50]	[0.5, 5]
DAG-4	30	0.6	0.7	[10^7, 10^8]	[5, 50]	[0.5, 5]

4.1. Performance Indicators

4.1.1. Comprehensive Cost. We set a weight space $\Omega = [\alpha_1, \alpha_2, \ldots, \alpha_{|\Omega|}]$, and given a weight value α_i, we first consider the optimization goals $T_A^{\alpha_i}$ and $E_A^{\alpha_i}$ for the time delay and energy consumption, respectively. According to the abovementioned description, the comprehensive cost function is as follows:

$$\mathrm{CC}_{\alpha_i} = \alpha_i \cdot T_A^{\alpha_i} + (1 - \alpha_i) \cdot E_A^{\alpha_i}. \tag{19}$$

Then, we calculate the average completion time (ACT), the average energy consumption (AEC), and the average comprehensive cost value (ACC), respectively.

$$\begin{aligned} \mathrm{AEC} &= \frac{1}{|\Omega|} \sum_{\alpha_i \in \Omega} E_A^{\alpha_i}, \\ \mathrm{ACT} &= \frac{1}{|\Omega|} \sum_{\alpha_i \in \Omega} T_A^{\alpha_i}, \\ \mathrm{ACC} &= \frac{1}{|\Omega|} \sum_{\alpha_i \in \Omega} CC_{\alpha_i}^{\alpha_i}. \end{aligned} \tag{20}$$

4.1.2. Network Usage. The network resource usage (NU) depends on the amount of data transferred between MT and BS. Thus, NU is defined as follows:

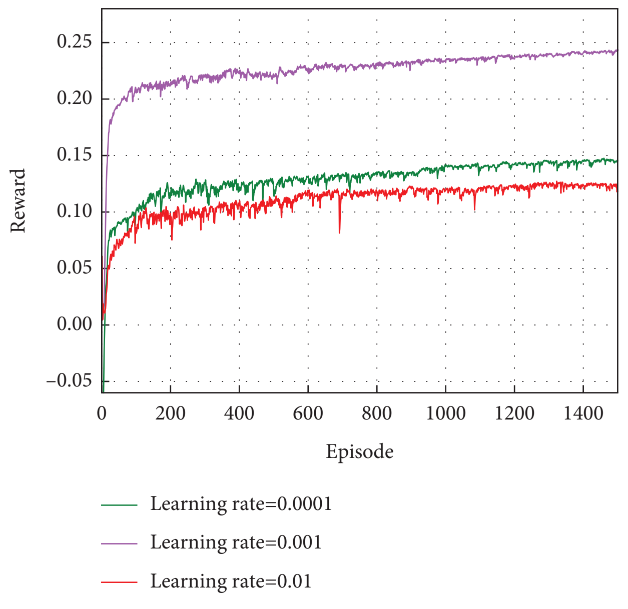
$$E_A = \frac{1}{|\Omega|} \sum_{\alpha_i \in \Omega} \frac{1}{\mathrm{ACT}} \sum_{i=1}^{N} \left(T_i^s \cdot d_i + T_i^r \cdot q_i\right) \cdot \mathrm{I}_{(a_i=1)}. \tag{21}$$

4.2. Parameter Analysis

4.2. Parameter Analysis. In this section, we compare the performance of the model with varied parameter values by taking the reward values as indicators. Figure 6 shows the influence of the learning rate in the optimizer on algorithm performance. It can be seen that systems with a too large or too small learning rate will not be rewarded very much. Meanwhile, considering the training efficiency and other factors, we set the learning rate to 0.001. The sampling efficiency in the case of a small batch size is relatively low, and a large batch size may lead to frequent selection of old samples in the experience pool. Therefore, the batch size is set to 500 in the present work.

4.3. Indicator Analysis. To gain insights into the proposed GMOM model, the following methods are implemented for comparison.

4.3.1. Greedy. Greedy decisions are made about subtasks based on estimates of the local and offloading computing completion time for each subtask.



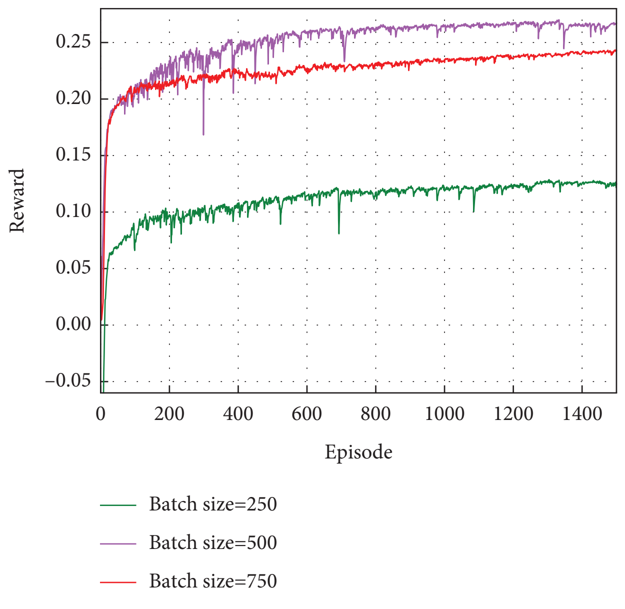
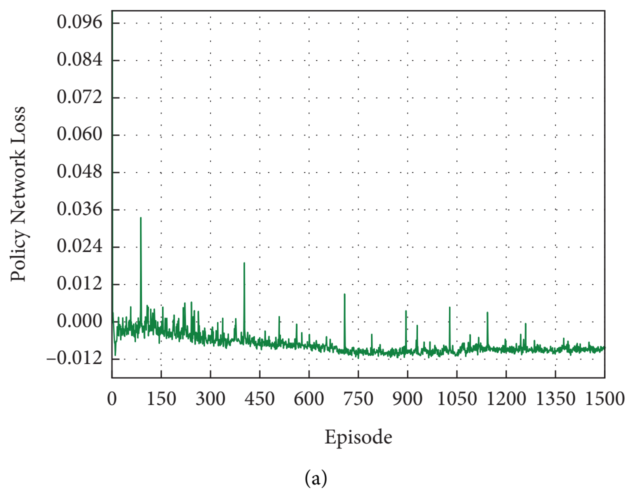


FIGURE 6: The reward under different parameters in DAG-3. (a) Rewards under different learning rates. (b) Rewards under different batch sizes.



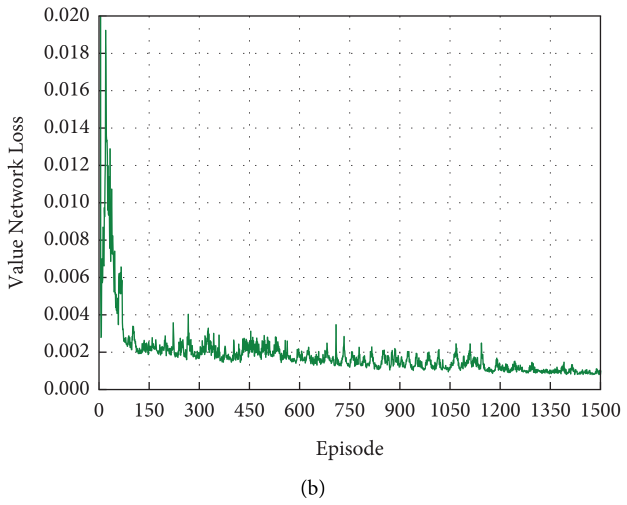


FIGURE 7: The loss of the policy network and the value network in the training process of GMOM. (a) The policy network loss. (b) The value network loss.

4.3.2. HEFT. All subtasks are prioritized according to the earliest completion time, and then, the subtasks with a higher priority are allocated with resources with the smallest estimated completion time for processing.

4.3.3. DQN. Long and short-term memory is combined with DQN to solve the offloading problem of dependent tasks in the heterogeneous MEC environment, with the goal of minimizing completion delay and energy consumption at the same time.

4.3.4. RL-DWS. To solve the problem of dynamic preference, multiobjective reinforcement learning with dynamic weights is proposed, and the change in the weight is measured by learning the Q value of multiobjectives.

Figure 7 shows the training results of two networks. As shown in Figure 6, the system reward increases sharply in the initial 200 episodes before the growth rate levels off, and this is also manifested in the loss value. Finally, the loss of the value network and policy network approaches 0, indicating that the algorithm in the present work has good convergence.

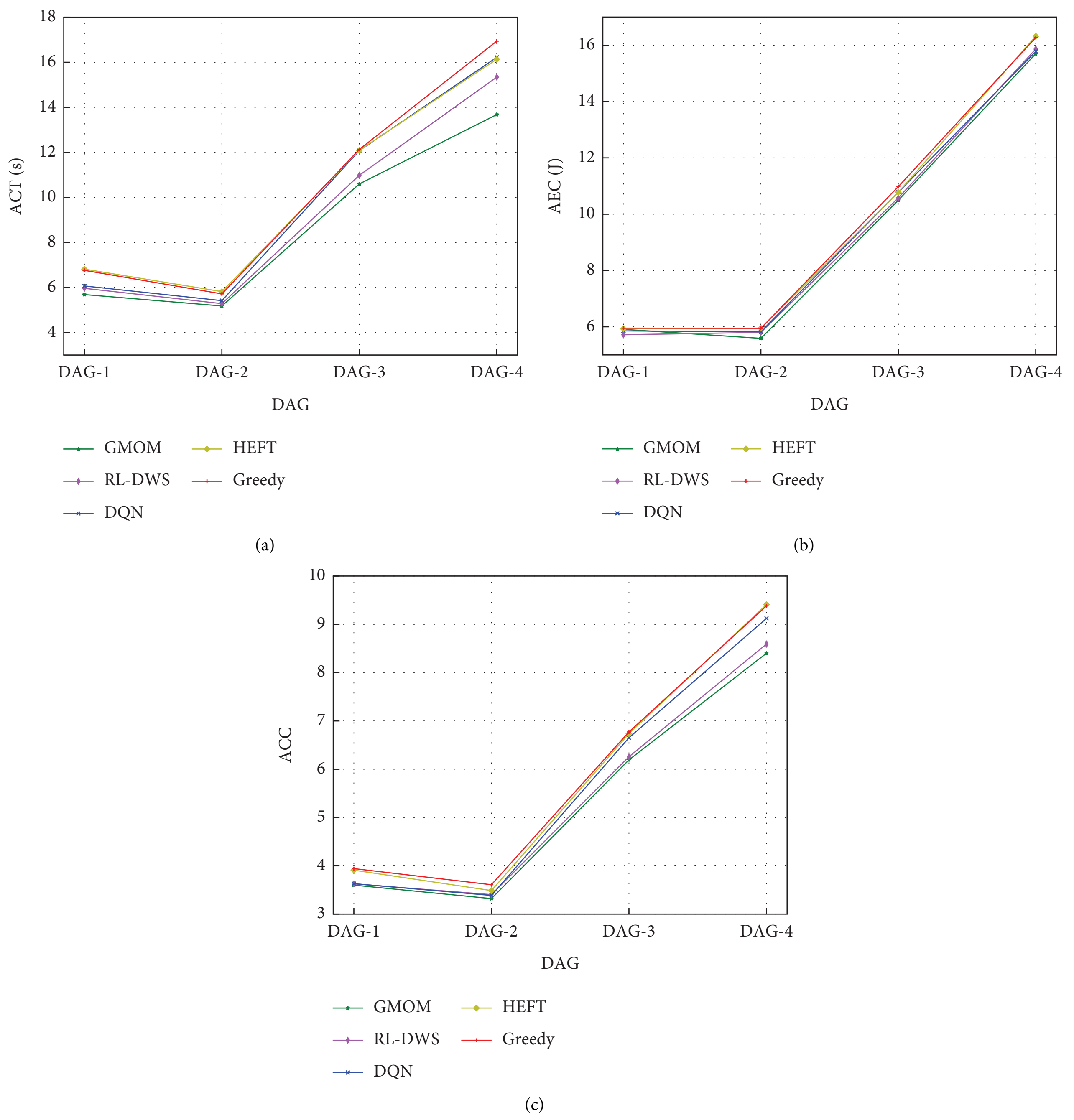


Figure 8: ACT, AEC, and ACC of all algorithms under different DAGs. (a) The ACT of all algorithms. (b) The AEC of all algorithms. (c) The ACC of all algorithms.

First, the transmission rate is set at 4.5 Mbps and the performance of different algorithms under different DAGs is compared, where $\Omega = [0, 0.2, 0.5, 0.8, 1]$. Figure 8 shows the corresponding ACT, AEC, and ACC values of different algorithms. As shown in Figure 8(a), our proposed algorithm can always obtain the minimum ACT, indicating that it obtains a smaller execution delay compared with other algorithms. Some algorithms, such as RL-DWS and DQN, perform well in terms of energy consumption, but poorly in terms of delay for DAG-4. HEFT and Greedy have similar performance in terms of delay and energy consumption for DAG-1 and DAG-3. The baseline algorithm underperforms in balancing is delay and energy consumption; our algorithm, however, shows good performance in terms of all the three evaluation indicators. In Figure 8(b), GMOM performs best in most DAGs (except for DAG-1). Although RL-DWS obtains the minimum AEC in DAG-1, its corresponding ACT is larger. In Figure 8(c), ACC of GMOM is also the smallest, indicating that GMOM can also perform best when comprehensively measuring delay and energy consumption.

Figure 9 represents NU values achieved by all algorithms discussed in our work. It shows that our algorithm gets a

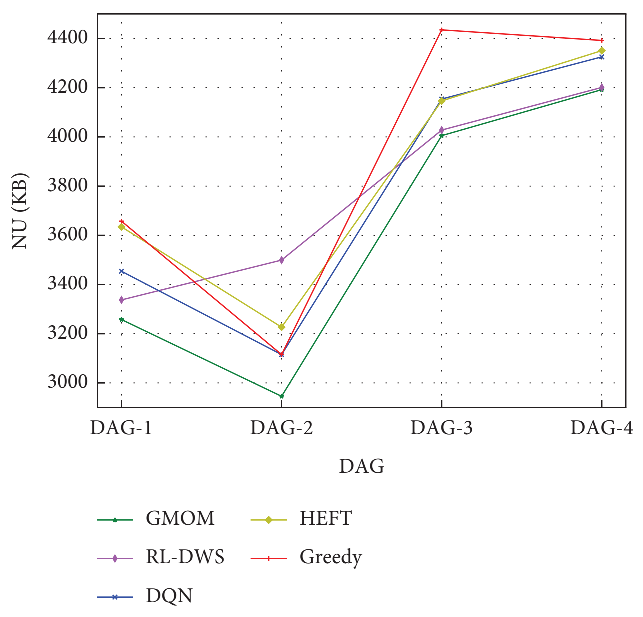


Figure 9: NU of all algorithms under different DAGs.

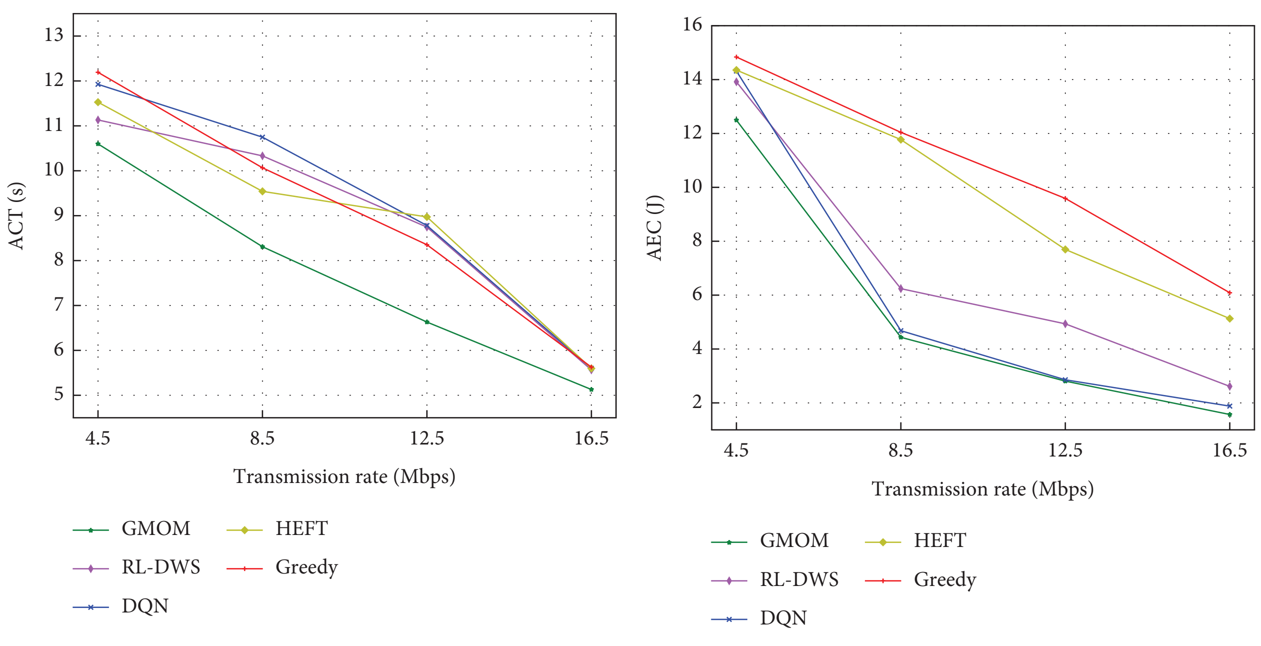
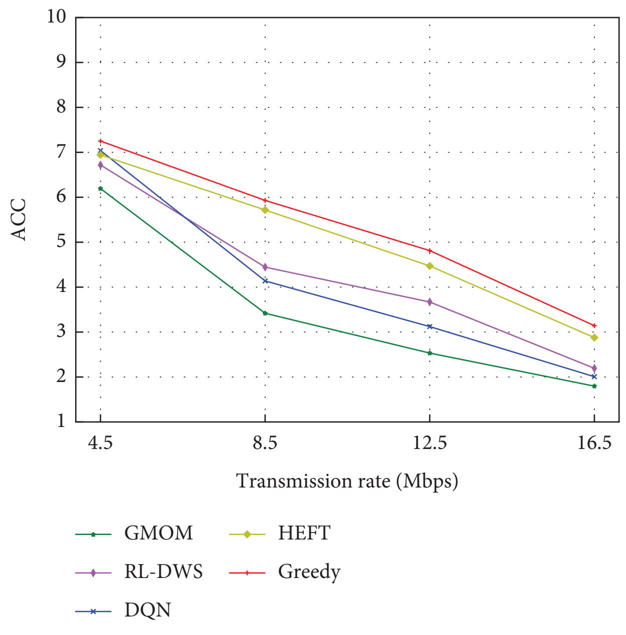


Figure 10: Continued.



(c)

Figure 10: ACT, AEC, and ACC of all algorithms at different transmission rates. (a) The ACT of all algorithms. (b) The AEC of all algorithms. (c) The ACC of all algorithms.

smaller NU value, indicating its better performance in task offloading between MT and BS than other baselines, which is conducive to the utilization of network resources.

Then, DAG 1–4 are tested with weight $\alpha = 0.2$ to compare the average performance of different algorithms at different transmission rates. At a small rate of transmission, offloading from MT to BS will suffer a high transmission delay; however, at a higher rate, the delay will decline. Our goal is to design an efficient algorithm to accommodate different transmission rates. In Figure 10(a), RL-DWS fails to learn an effective strategy, and its performance is worse than that of HEFT when the rate increases from 4.5 Mbps to 8.5 Mbps. On the contrary, our algorithm has better adaptability than all baselines and approaches the optimal solution at all rates. In Figures 10(b) and 10(c), DQN is superior to RL-DWS on AEC, but the results in Figure 10(a) are opposite. Meanwhile, Figure 10(c) shows that ACC of all algorithms decreases as the transmission rate increases. This is because as the communication cost declines the transmission rate increases, offloading tasks to edge servers may be beneficial.

The weight reflects the importance of delay under a particular offloading strategy. At a small weight value, the offloading strategy should be selected to reduce the delay. Figure 11 shows the trend of changes in ACT and AEC as the weight of the delay increases. Overall, ACT decreases while AEC increases, because as the weight increases, the number of subtasks selecting local computing decreases, while the number of subtasks opting for offloading computing increases. This is mainly because MT has a much lower computing capacity than BS. However, offloading more subtasks to BS will lead to higher energy consumption. Therefore, the time delay and energy consumption can be effectively balanced through edge-end collaborative processing of tasks.

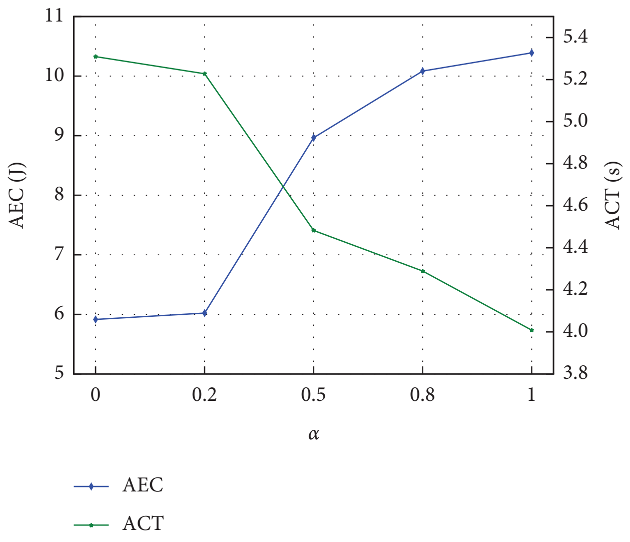


Figure 11: ACT and AEC with different weights.

5. Conclusion

In the present work, the offloading problem of dependent tasks in the mobile edge computing environment is studied to balance delay and energy consumption. We propose a graph mapping offloading model based on deep reinforcement learning; specifically, the offloading problem is modeled as an MDP, and RNN is combined to approximate the policy and value function of MDP, and it is combined with an encoding-decoding framework that introduces the attention mechanism to train the model with a popular proximal policy optimization algorithm. Simulation experiments show that the proposed algorithm has better stability and convergence and can obtain an approximately optimal solution.

In order to meet the expected large number of demand services, the research of future generation wireless network (6G) has been initiated, which is expected to improve the enhanced broadband, massive access, and low latency service capability of the 5G wireless network, which is beneficial for the mobile edge network [29]. However, 6G networks tend to be multidimensional, ultradense, and heterogeneous, so artificial intelligence (AI), especially machine learning (ML), is emerging as a solution for intelligent network orchestration and management. For example, intelligent or intelligent spectrum management can be achieved using deep reinforcement learning, especially for random state measurement problems, which also has potential.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was funded by the National Natural Science Foundation of China (Grant no. U1603261) and the Natural Science Foundation Program of Xinjiang Province, China (Program no. 2016D01A080).

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: the communication perspective,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, “On multi-access edge computing: a survey of the emerging 5G network edge cloud architecture and orchestration,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[3] R. D. Yates and S. Kaul, “Real-time status updating: multiple sources,” in *Proceedings of the 2012 IEEE International Symposium on Information Theory Proceedings*, pp. 2666–2670, Cambridge, MA, USA, August 2012.

[4] P. Mach and Z. Becvar, “Mobile edge computing: a survey on architecture and computation offloading,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[5] A. Vatankhah and S. Babaie, “An optimized Bidding-based coverage improvement algorithm for hybrid wireless sensor networks,” *Computers & Electrical Engineering*, vol. 65, no. 4, pp. 1–17, 2018.

[6] L. Singh and S. Singh, “A survey of workflow scheduling algorithms and research issues,” *International Journal of Computer Application*, vol. 74, no. 15, pp. 21–28, 2013.

[7] S. Sundar and B. Liang, “Offloading dependent tasks with communication delay and deadline constraint,” in *Proceedings of the IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 37–45, Honolulu, HI, USA, April 2018.

[8] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, “Offloading dependent tasks in mobile edge computing with service caching,” in *Proceedings of the IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 1997–2006, Toronto, ON, Canada, August 2020.

[9] H. Peng, W. S. Wen, M. L. Tseng, and L. L. Li, “Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment,” *Applied Soft Computing*, vol. 80, pp. 534–545, 2019.

[10] A. A. Al-Habob, O. A. Dobre, A. G. Armada, and S. Muhaidat, “Task scheduling for mobile edge computing using genetic algorithm and conflict graphs,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 8805–8819, 2020.

[11] S. S. Mousavi, M. Schukat, and E. Howley, “Deep reinforcement learning: an overview,” 2018, https://arxiv.org/abs/1701.07274.

[12] D. Silver, T. Hubert, J. Schrittwieser et al., “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[13] Y. Zhan, S. Guo, P. Li, and J. Zhang, “A deep reinforcement learning based offloading game in edge computing,” *IEEE Transactions on Computers*, vol. 69, no. 6, pp. 883–893, 2020.

[14] M. Tang and V. W. Wong, “Deep reinforcement learning for task offloading in mobile edge computing systems,” *IEEE Transactions on Mobile Computing*, vol. 21, no. 6, pp. 1985–1997, 2022.

[15] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, “Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning,” *IEEE Communications Magazine*, vol. 57, no. 5, pp. 64–69, 2019.

[16] J. Zou, T. Hao, C. Yu, and H. Jin, “A3c-do:A regional resource scheduling framework based on deep reinforcement learning inedge scenario,” *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 228–239, 2021.

[17] J. Zhang, J. Du, Y. Shen, and J. Wang, “Dynamic computation offloading with energy harvesting devices: a hybrid-decision-based deep reinforcement learning approach,” *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9303–9317, 2020.

[18] F. Wu, Q. Wu, and Y. Tan, “Workflow scheduling in cloud: a survey,” *The Journal of Supercomputing*, vol. 71, no. 9, pp. 3373–3418, 2015.

[19] F. Song, H. Xing, X. Wang, S. Luo, P. Dai, and K. Li, “Offloading dependent tasks in multi-access edge computing: a multi-objective reinforcement learning approach,” *Future Generation Computer Systems*, vol. 128, pp. 333–348, 2022.

[20] F. Song, H. Xing, S. Luo, D. Zhan, P. Dai, and R. Qu, “A multi objective computation offloading algorithm for mobile edge computing,” *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8780–8799, 2020.

[21] W. Zhan, C. Luo, G. Min, C. Wang, Q. Zhu, and H. Duan, “Mobility-aware multi-user offloading optimization for mobile edge computing,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3341–3356, 2020.

[22] H. Cao and J. Cai, “Distributed multi-user computation offloading for cloudlet-based mobile cloud computing: a game-theoreticmachine learning approach,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 752–764, 2018.

[23] Y.-K. Kwok and I. Ahmad, “Static scheduling algorithms for allocating directed task graphs to multi-processors,” *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.

[24] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of the 3rd International Conference on Learning Representations*, ICLR, San Diego, CA, USA, May 2015.

[25] Z. W. Wang, *Research on End-To-End Deep Reinforcement Learning Control for Intelligent Vehicle Based on PPO Algorithm*, Jilin university, Changchun, China, 2021.

[26] J. Du, C. Jiang, J. Wang, Y. Ren, and M. Debbah, "Machine learning for 6G wireless networks: carrying forward enhanced bandwidth, massive access, and ultrareliable/low-latency service," *IEEE Vehicular Technology Magazine*, vol. 15, no. 4, pp. 122–134, 2020.

[27] Z. Zhang, Y. Xiao, Z. Ma et al., "6G wireless networks: vision, requirements, architecture, and key technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, 2019.

[28] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm forheterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2014.

[29] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for distributed dynamic spectrum access," *IEEE Transactions on Wireless Communications*, vol. 18, no. 1, pp. 310–323, 2019.